

## PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-014826

(43)Date of publication of application : 18.01.2002

(51)Int.Cl.

G06F 9/45

(21)Application number : 2000-196871

(71)Applicant : TOSHIBA CORP

(22)Date of filing : 29.06.2000

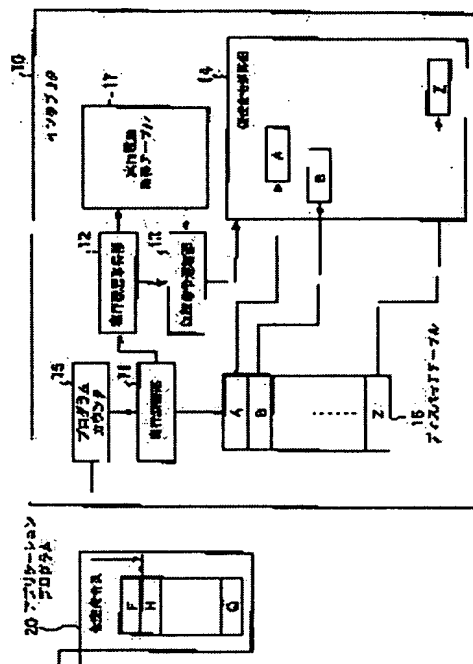
(72)Inventor : SAKAI RYUJI

## (54) INTERPRETER AND METHOD FOR EXECUTING ITS PROGRAM

## (57)Abstract:

PROBLEM TO BE SOLVED: To provide an interpreter capable of realizing acceleration equivalent to that of JIT compiler with a simpler mechanism without scarifying the starting time of a program.

SOLUTION: An execution control part 11 obtains, from a dispatch table 16, the entry address of a virtual instruction processing part 14 for processing a virtual instruction 21 indicated by a program counter 15 allows the address to jump, and notifies an execution history obtaining part 12 of which virtual instruction is the target for execution control. On the other hand, the execution history obtaining part 12 makes a pair of the virtual instruction and the previously communicated virtual instruction, and monitors the execution frequency by counting the number of times of appearance by using an execution history obtaining table 17, and notifies a virtual instruction connecting part 13 of the connection of the pair whose execution frequency is high as one virtual instruction. Then, the virtual instruction connecting part 13 executes the reconstitution of a virtual instruction series in order to connect the communicated pair as one virtual instruction.



## LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's]



## 【特許請求の範囲】

【請求項1】 複数の仮想命令からなる一連の仮想命令列の実行履歴を取得する実行履歴取得手段と、前記実行履歴取得手段により取得された実行履歴から実行頻度の高い前記一連の仮想命令列を検出する検出手段と、前記検出手段により検出された前記一連の仮想命令列を一括処理する処理部を新たに生成して未定義の空き仮想命令を割り当てるとともに、その割り当てた空き仮想命令として解釈実行されるように前記一連の仮想命令列を再構成する仮想命令連結手段と、を具備することを特徴とするインタプリタ。

【請求項2】 前記仮想命令連結手段は、新たに生成して未定義の空き仮想命令を割り当てた処理部が予め定められた条件を満たすときに、その処理部をアドレス呼び出し用の仮想命令から呼び出すべく解釈実行されるように前記一連の仮想命令列をさらに再構成する手段を有することを特徴とする請求項1記載のインタプリタ。

【請求項3】 複数の仮想命令からなる一連の仮想命令列の実行履歴を取得する実行履歴取得手段と、前記実行履歴取得手段により取得された実行履歴から実行頻度の高い前記一連の仮想命令列を実行中に検出する検出手段と、前記検出手段により検出された前記一連の仮想命令列を一括処理する処理部を新たに生成するとともに、その生成した処理部をアドレス呼び出し用の仮想命令から呼び出すべく解釈実行されるように前記一連の仮想命令列を再構成する仮想命令連結手段と、を具備することを特徴とするインタプリタ。

【請求項4】 前記アドレス呼び出し用の仮想命令は、未定義の空き仮想命令を用いて設けられるものであることを特徴とする請求項2または3記載のインタプリタ。

【請求項5】 最適化の対象を限定する手段を有することを特徴とする請求項1、2、3または4記載のインタプリタ。

【請求項6】 仮想命令と処理部とを対応づけるディスパッチテーブルを最適化の対象別に備えて、それぞれに固有の最適化を行うことを特徴とする請求項1、2、3または4記載のインタプリタ。

【請求項7】 複数の仮想命令からなる一連の仮想命令列の実行履歴を取得し、この取得した実行履歴から実行頻度の高い前記一連の仮想命令列を検出し、この検出した前記一連の仮想命令列を一括処理する処理部を新たに生成して未定義の空き仮想命令を割り当てるとともに、その割り当てた空き仮想命令として解釈実行されるように前記一連の仮想命令列を再構成することを特徴とするインタプリタのプログラム実行方法。

【請求項8】 新たに生成して未定義の空き仮想命令を割り当てた処理部が予め定められた条件を満たすとき

に、その処理部をアドレス呼び出し用の仮想命令から呼び出すべく解釈実行されるように前記一連の仮想命令列をさらに再構成することを特徴とする請求項7記載のインタプリタのプログラム実行方法。

【請求項9】 複数の仮想命令からなる一連の仮想命令列の実行履歴を取得し、

この取得した実行履歴から実行頻度の高い前記一連の仮想命令列を検出し、

この検出した前記一連の仮想命令列を一括処理する処理部を新たに生成するとともに、その生成した処理部をアドレス呼び出し用の仮想命令から呼び出すべく解釈実行されるように前記一連の仮想命令列を再構成することを特徴とするインタプリタのプログラム実行方法。

## 【発明の詳細な説明】

## 【0001】

【発明の属する技術分野】この発明は、仮想命令を用いて記述されたプログラムを解釈実行するインタプリタおよびそのプログラム実行方法に係り、特に、プログラムの起動時間を犠牲にすることなく、JITコンパイラより単純な仕組みで同等の高速化を実現するインタプリタおよびそのプログラム実行方法に関する。

## 【0002】

【従来の技術】インタプリタとは、仮想的な命令をソフトウェアで解釈し、実行するプログラムのことである。インタプリタは、命令を解釈し実行する一つのループによって実装される。このループの中には、仮想命令をメモリからロードして命令の種類を判別し、その命令に相当する処理へ分岐する命令ディスパッチ処理の部分と各命令毎に命令の機能を実行する部分とがある。

【0003】インタプリタによるプログラムの実行は、同等の処理をターゲットのマイクロプロセッサの機械語命令にコンパイルして動作するプログラムと比較すると、数倍から十数倍遅くなる。そこで、インタプリタによる実行を少しでも高速にするために、実行する仮想命令を、予め決められた別の命令に変更したり、複数の仮想命令を同等の処理を行う一つの仮想命令に置き換えることが行われている。また、一連の仮想命令列を、プログラム実行時に機械語命令列にコンパイルしてから実行するJIT(Just In Time)コンパイルという手法も存在する。

## 【0004】

【発明が解決しようとする課題】ところで、この仮想命令を別の命令に書き換えながら実行する方法は、予め特定の仮想命令や仮想命令のペア、あるいは仮想命令のシーケンスを、特定の仮想命令(空き仮想命令を使うこともある)に置き換えるための規則と仕組みとをインタプリタに与えておく必要があるので、仮想命令の定義できる範囲でしか書き換え規則を定義できないので、高速化の観点からは、それほど大きな効果がなかった。

【0005】一方、JITコンパイルによる高速化は、

仮想命令列より機械語命令列の方がサイズが大きくなるため、多量のメモリを必要とするので、メモリサイズの小さい組み込み機器などでは使えないことが多い。また、各処理単位の最初の実行時に J I T コンパイルする場合、プログラムの起動時間が長くなってしまうために J I T コンパイラの使用をやめてしまうケースも多い。

【0006】また、メモリサイズの小さなシステムでは、全ての処理を J I T コンパイルするのではなく、実行頻度の高い処理のみを J I T コンパイルすればよいが、実行頻度を求めて J I T コンパイルする仕組みが、多くのメモリを使用してしまうという問題もある。

【0007】この発明はこのような事情を考慮してなされたものであり、プログラムの起動時間を犠牲にすることなく、J I T コンパイラより単純な仕組みで同等の高速化を実現するインタプリタおよびそのプログラム実行方法を提供することを目的とする。

【0008】

【課題を解決するための手段】前述した目的を達成するために、この発明のインタプリタは、拡張性の確保等のために一般的に用意される空き仮想命令を利用して、プログラムの実行中に実行頻度の高い仮想命令列を動的に 1 つの仮想命令として再構成していくようにしたものであり、そのために、複数の仮想命令からなる一連の仮想命令列の実行履歴を取得する実行履歴取得手段と、前記実行履歴取得手段により取得された実行履歴から実行頻度の高い前記一連の仮想命令列を検出する検出手段と、前記検出手段により検出された前記一連の仮想命令列を一括処理する処理部を新たに生成して未定義の空き仮想命令を割り当てるとともに、その割り当てた空き仮想命令として解釈実行されるように前記一連の仮想命令列を再構成する仮想命令連結手段とを具備することを特徴とする。

【0009】この発明のインタプリタにおいては、実行頻度の高い仮想命令列について、その実行に必要となる n 回のループを 1 回のループで済むように動的に再構成を行うことにより、プログラムの起動時間を犠牲にすることなく、J I T コンパイラより単純な仕組みで同等の高速化を実現することを可能とする。

【0010】また、この発明のインタプリタは、前記仮想命令連結手段が、新たに生成して未定義の空き仮想命令を割り当てた処理部が予め定められた条件を満たすときに、その処理部をアドレス呼び出し用の仮想命令から呼び出すべく解釈実行されるように前記一連の仮想命令列をさらに再構成する手段を有することを特徴とする。

【0011】この発明のインタプリタにおいては、有限である空き仮想命令を有効に利用することを可能とす。

【0012】また、この発明のインタプリタは、前記アドレス呼び出し用の仮想命令が、未定義の空き仮想命令を用いて設けられるものであることを特徴とする。

【0013】この発明のインタプリタにおいては、アド

レス呼び出し用の仮想命令が予め定義されていない場合であっても、同手法での最適化を行うことを可能とする。

【0014】また、この発明のインタプリタは、最適化の対象を限定する手段を有することを特徴とする。

【0015】この発明のインタプリタにおいては、たとえばプログラムの実行計画などに対応させて最適化の効果を得やすくすることを可能とする。

【0016】また、この発明のインタプリタは、仮想命令と処理部とを対応づけるディスパッチテーブルを最適化の対象別に備えて、それぞれに固有の最適化を行うことを特徴とする。

【0017】この発明のインタプリタにおいては、たとえばメソッド、関数または処理単位等ごとに最適化を行うことを可能とする。

【0018】

【発明の実施の形態】以下、図面を参照してこの発明の一実施形態を説明する。

【0019】図 1 は、この発明の実施形態に係るインタプリタのプログラム実行方法を説明するための概念図であり、インタプリタ 10 は、実行制御部 11、実行履歴取得部 12、仮想命令連結部 13 および仮想命令処理部 14 の各処理部と、プログラムカウンタ 15、ディスパッチテーブル 16 および実行履歴取得テーブル 17 の各データ部とにより、アプリケーションプログラム 20 の仮想命令列 21 を解釈実行しながらその最適化も動的に行っていく。

【0020】なお、この図 1 に示すインタプリタ 10 およびアプリケーションプログラム 20 は、ハードディスク等の外部メモリからコンピュータの主メモリにロードされたものであり、たとえば後述する仮想命令列 21 の書き換え等は、あくまで主メモリ上だけで行われるものである。

【0021】実行制御部 11 は、プログラムカウンタ 15 が指し示す仮想命令 21 をロードして、その仮想命令 21 を処理するための仮想命令処理部 14 のエントリアドレスをディスパッチテーブル 16 から取得し、その取得した仮想命令処理部 14 のエントリアドレスに制御を移す（ジャンプする）。

【0022】このディスパッチテーブル 16 は、各仮想命令と仮想命令処理部 14 のエントリアドレスとを対応づけるためのものであるが、図 2 に示すように、予め定義された既存の仮想命令のエントリアドレスを格納する領域に加え、拡張性を確保する等のために、新設の仮想命令のエントリアドレスを格納するための空き領域を備えている。また、このインタプリタ 10 では、その空き領域で新設される仮想命令の一つを、後述するネイティブ手続き命令（オペランドとしてエントリアドレスを受け取り、そのエントリアドレスにジャンプする処理を実行する）として予約する。

【0023】また、このとき、実行制御部11は、どの仮想命令について実行制御したのかを実行履歴取得部12に通知する。

【0024】一方、この通知を受けた実行履歴取得部12は、この通知された仮想命令と一つ前に通知された仮想命令とをペアにして、その出現回数を実行履歴取得テーブル17を用いてカウントする。図3は、実行履歴取得テーブル17の一例を示す図であり、たとえば今回通知された仮想命令がHでその一つ前に通知された仮想命令がFであれば、実行履歴取得部12は、F行H列（F，H）の値をインクリメントし、逆に、今回通知された仮想命令がFでその一つ前に通知された仮想命令がHであれば、実行履歴取得部12は、H行F列（H，F）の値をインクリメントすることにより、各ペアの出現回数をカウントする。

【0025】実行履歴取得部12は、しきい値Sとサンプル期間Tとをもち、また、このしきい値Sを制御するための値s1，s2をもっている（s1<s2）。そして、実行履歴取得部12は、まず、しきい値Sをs1で初期化し、いずれかのペアの出現回数がしきい値Sを越え、これらを1つの仮想命令として連結すべくその旨を仮想命令連結部13に通知する。なお、実行履歴取得部12は、サンプル期間Tが経過するごとに、しきい値Sにs1を加えていき、しきい値S>s2となったら実行履歴取得テーブル17のすべてのカウント値を0で初期化するとともに、しきい値Sをs1で初期化し直す。これにより、仮想命令の連結をより適応的に行なうことを実現する。

【0026】そして、仮想命令連結部13は、実行履歴取得部12から通知された仮想命令のペアを一つの仮想命令に連結すべく仮想命令列の再構成を実行する。以下、図4乃至図7を参照して、この仮想命令連結の一例を説明する。

【0027】いま、仮想命令Gと仮想命令Rのペア（G，R）の出現回数がしきい値Sを越え、実行履歴取得部12から仮想命令連結部13に対してこれらの連結が通知されたものと想定する。図4は、ペア（G，R）の連結が行われる前の状態を示す図である。

【0028】図4に示すように、連結前の状態では、アプリケーションプログラム20の仮想命令列21にはペア（G，R）が多数点在しており、一方、インタプリタ10のディスパッチテーブル16には、仮想命令Gおよび仮想命令Rともに仮想命令処理部14の正規の処理部Gおよび処理部Rのエントリアドレスが格納されている。

【0029】ペア（G，R）を連結する際、図5に示すように、仮想命令連結部13は、まず、仮想命令Gの次が仮想命令Rかどうかをチェックする処理部C（G，R）と、仮想命令Gと仮想命令Rとを連結した処理部J（G，R）と命令書き換えのための処理部W（G，R）

とを生成する処理部G（G，R）とを仮想命令処理部14に生成し、ディスパッチテーブル16に格納する仮想命令Gのエントリアドレスを処理部Gのエントリアドレスから処理部C（G，R）のエントリアドレスに書き換える。図5では、この処理を完了した時点での状態を示している。

【0030】次に、この状態でプログラムカウンタ12によりペア（G，R）の仮想命令Gが指し示されると、実行制御部11によってディスパッチテーブル16が参照され、書き換えられた処理部C（G，R）へのジャンプが行われることになる。

【0031】処理部C（G，R）は、次の仮想命令が実行制御部11からフェッチされ、その仮想命令がRであれば処理部G（G，R）に、そうでなければ、処理部Gにジャンプする（処理部Gにジャンプする代わりに処理部Gのコピーを結合しておいてもよい）ものであり、ここでは、次が仮想命令Rであるため、処理部G（G，R）にジャンプする。

【0032】一方、図6に示すように、処理部G（G，R）は、まず、処理部Gと処理部Rとを連結した処理部J（G，R）を仮想命令処理部14に生成し、ディスパッチテーブル16の空き仮想命令のプールから1つの仮想命令を取得して割り当てるとともに（これをJ（G，R）とする）、このペアも実行履歴取得部12のカウント対象とすべく実行履歴取得テーブル17の行列に加える。

【0033】次に、処理部G（G，R）は、インタプリタされているアプリケーションプログラム20の仮想命令ペア（G，R）を書き換えるための処理部W（G，R）を仮想命令処理部14に生成し、処理部C（G，R）から処理部G（G，R）への分岐を処理部W（G，R）への分岐に書き換える。

【0034】最後に、処理部G（G，R）は、処理部W（G，R）へジャンプする（処理部W（G，R）は、処理部G，処理部Rを数値パラメータとして与えられれば実現できる定型的な処理なので、実行前に用意してもよい）。図6では、この処理を完了した時点での状態を示している。

【0035】また、図7に示すように、ジャンプ先の処理部W（G，R）では、インタプリタされている仮想命令列21のカレントのプログラムカウンタ12が指し示している仮想命令GをJ（G，R）に書き換え、また、次の仮想命令Rをオペランドに書き換え、処理部J（G，R）にジャンプする。そして、処理部J（G，R）にて、仮想命令ペア（G，R）が一括して実行される。図7では、この処理を完了した時点での状態を示している。

【0036】以降、仮想命令列21の仮想命令ペア（G，R）が現れる度に、処理部C（G，R）から処理部W（G，R）へのジャンプが行われ、仮想命令列21

の書き換えと仮想命令ペア (G, R) の一括実行とが繰り返され、すべての仮想命令ペア (G, R) がいずれ J (G, R) に書き換えられることになる。また、仮想命令 G の次に仮想命令 R 以外の仮想命令がくる場合は、処理部 C (G, R) から処理部 G にジャンプして何ら問題なく処理される。さらに、その書き換え完了後であれば、仮想命令 G が重複するたとえば仮想命令ペア (G, S) についても、ディスパッチテーブル 16 に格納する仮想命令 G のエントリアドレスを処理部 J (G, R) のエントリアドレスから処理部 J (G, S) のエントリアドレスに置き換えつつ、同様の処理を行うことが可能である。

【0037】このように、このインタプリタのプログラム実行方法によれば、実行頻度の高い仮想命令列について、その実行に必要となる n 回のループを 1 回のループで済むように動的に再構成を行うことにより、プログラムの起動時間を犠牲にすることなく、JIT コンパイラより単純な仕組みで同等の高速化を実現することを可能とする。

【0038】また、この連結によって生成された仮想命令 J (G, R) も、他の仮想命令とのペアの出現回数がカウントされ、アプリケーションプログラム 20 の実行状況によっては、さらに他の仮想命令との連結が進められることになる。

【0039】ところで、このように連結を進めていくと、実行頻度の高い部分の仮想命令列はいずれ 1 つの仮想命令となる。そこで、仮想命令連結部 13 は、1 つの仮想命令に連結しようとする元の仮想命令の数が、所定のしきい値を越えるか、あるいは基本ブロック全域となる場合に、その実行効率がより高速になるように、連結した処理単位に最適化をかける。ここでいう基本ブロックとは、プログラムを分岐命令と分岐命令の分岐先とで分割したときにできる必ずシーケンシャルに実行される各部分をいう。

【0040】このとき、仮想命令連結部 13 は、この仮想命令の処理部が基本ブロックの全域になっていれば、隣接する基本ブロックを調べ、隣接する基本ブロックも 1 つの仮想命令であって、かつ、これら基本ブロックの連続領域について、入口と出口とがそれぞれ 1 つの場合に、これら複数の基本ブロックの処理を一つの仮想命令として、この全体の処理について最適化した処理部を生成する。最適化された処理部は、もはやオペランド部の情報を必要としないので、連結された仮想命令のオペランド部の長さがターゲットのマイクロプロセッサの手続きのアドレス表現に必要な長さ以上になった場合に、この仮想命令を、前述したネイティブ手続き呼出命令 (オペランドとしてエントリアドレスを受け取り、そのエントリアドレスにジャンプする処理を実行する) に書き換える。以下、図 8 および図 9 を参照して、この仮想命令書き換えの一例を説明する。

【0041】いま、連結により新たに生成された仮想命令 J 1 ~ J 4 が存在し、かつ、仮想命令 J 1 と仮想命令 J 2、および仮想命令 J 3 と仮想命令 J 4 をさらに連結するものとする。そして、双方ともに、ネイティブ手続き呼出命令に書き換えるための条件を満たすものとする。図 8 は、この書き換え前の状態を示す図である。

【0042】図 8 に示すように、書き換え前の状態では、アプリケーションプログラム 20 の仮想命令列 21 にはペア (J 1, J 2) とペア (J 3, J 4) とが点においており、一方、インタプリタ 10 のディスパッチテーブル 16 には、仮想命令 J 1 ~ J 4 それぞれに仮想命令処理部 14 の新設の処理部 J 1 ~ J 4 のエントリアドレスが格納されている。

【0043】そして、図 9 に示すように、仮想命令連結部 13 は、まず、仮想命令 J 1 ~ J 2 を連結した処理部 J (J 1 + J 2) と仮想命令 J 3 ~ J 4 を連結した処理部 J (J 3 + J 4) とを仮想命令処理部 14 に生成し、次いで、仮想命令列 21 に点するペア (J 1, J 2) をオペランドとして処理部 J (J 1 + J 2) のエントリアドレスをもったネイティブ手続き呼出命令に書き換え、また、仮想命令列 21 に点するペア (J 3, J 4) をオペランドとして処理部 J (J 3 + J 4) のエントリアドレスをもったネイティブ手続き呼出命令に書き換える。図 9 では、この処理を完了した時点での状態を示している。

【0044】これにより、連結された処理単位のさらなる高速化が図られ、また、J 1 ~ J 4 の実行頻度が低下することにより、結果として有限である空き仮想命令を有効利用することが可能となる。

【0045】次に、図 10 および図 11 を参照して、このインタプリタ 10 のプログラム実行方法の動作手順を説明する。

【0046】このインタプリタ 10 は、実行制御部 11 が、プログラムカウンタ 15 が指し示す仮想命令 21 をロードし (図 10 のステップ A 1)、同時に、実行履歴取得部 12 が、仮想命令列の実行履歴を取得する (図 10 のステップ A 2)。

【0047】また、実行履歴取得部 12 は、出現回数がしきい値を越える仮想命令列が存在しないかを監視し (図 10 のステップ A 3)、しきい値を越える仮想命令列の存在を検出すれば (図 10 のステップ A 3 の YES)、その仮想命令列の連結を仮想命令連結部 13 に実行させる (図 10 のステップ A 4)。

【0048】一方、仮想命令連結部 13 は、連結すべき仮想命令列のもとの仮想命令数がしきい値を越えるか、あるいは基本ブロックとなっているかを判断し (図 11 のステップ B 1)、もとの仮想命令数がしきい値を越えるか、あるいは基本ブロックとなっていた場合 (図 11 のステップ B 1 の YES)、この連結すべき仮想命令列をネイティブ手続き呼出命令とすべく仮想命令列の再構

成を実行する(図11のステップB2)。

【0049】また、もとの仮想命令数がしきい値を越えず、かつ基本ブロックともなっていない場合(図11のステップB1のN0)、まず、仮想命令連結部13は、空き仮想命令の取得を試み(図11のステップB3)、もし取得できなければ(図11のステップB4のN0)、実行頻度の最も低い空き仮想命令を解放する(ステップB5)。この解放は、その空き仮想命令のために書き換えた仮想命令列21およびディスパッチテーブル16を元に戻すことによって実施する。その後、仮想命令連結部13は、連結すべき仮想命令列をその取得した空き仮想命令とすべく仮想命令列の再構成を実行する(図11のステップB6)。

【0050】そして、実行制御部11は、ディスパッチテーブル16から取得した仮想命令処理部14のエントリアドレスにジャンプし(図10のステップA5)、その仮想命令のサイズ分だけプログラムカウンタ12をインクリメントした後(図10ステップA6)、上記の処理を繰り返す。

【0051】このように、このインタプリタのプログラム実行方法によれば、実行頻度の高い仮想命令列について、その実行に必要となるn回のループを1回のループで済むように動的に再構成を行うことにより、プログラムの起動時間を犠牲にすることなく、JITコンパイラより単純な仕組みで同等の高速化を実現することを可能とする。

【0052】なお、前述の実施形態では、仮想命令列の連結を、まず空き仮想命令を用いて実行することを原則とし、状況によって、ネイティブ手続き呼び出し命令を用いる例を説明したが、リソース環境等によっては、当初よりネイティブ手続き呼び出し命令を用いて実行しても構わない。

【0053】また、前述の実施形態では、ネイティブ手続き呼び出し命令を空き仮想命令を用いて設ける例を説明したが、アドレス呼び出し用の仮想命令が予め定義されている場合には、その既存の仮想命令を用いれば良い。

【0054】ところで、仮想命令数が少なく、空き仮想命令が十分存在する場合は、小さなループのようなある程度まとまった部分の処理が1つの仮想命令となったリ、ネイティブ手続き呼出命令となって、JITコンパイラと同等の性能を得ることが可能であるが、空き仮想命令が十分存在しないときには、十分な効果が得られないおそれがある。そこで、実行履歴を採取して最適化する対象を、そのとき高速化したい手続きに限定する機能をさらに備えれば、たとえばプログラムの実行計画などに対応させて最適化の効果を得やすくすることが可能となる。

【0055】さらに、メソッド、関数または処理単位等、最適化の対象ごとに異なるディスパッチテーブル1

6を用いれば、このメソッド、関数または処理単位等ごとに固有の仮想命令連結を行うことが可能となる。

【0056】

【発明の効果】以上詳述したように、この発明のインタプリタによれば、拡張性の確保等のために一般的に用意される空き仮想命令を利用して、プログラムの実行中に実行頻度の高い仮想命令列を動的に1つの仮想命令として再構成していくようにしたことから、実行頻度の高い仮想命令列について、その実行に必要となるn回のループを1回のループで済むように動的に再構成を行うことにより、プログラムの起動時間を犠牲にすることなく、JITコンパイラより単純な仕組みで同等の高速化を実現することを可能とする。

【図面の簡単な説明】

【図1】この発明の実施形態に係るインタプリタのプログラム実行方法を説明するための概念図。

【図2】同実施形態のインタプリタが備えるディスパッチテーブルの構成を示す図。

【図3】同実施形態のインタプリタが備える実行履歴取得テーブルの構成を示す図。

【図4】同実施形態のインタプリタが実行する仮想命令連結の一例を説明するための第1の図。

【図5】同実施形態のインタプリタが実行する仮想命令連結の一例を説明するための第2の図。

【図6】同実施形態のインタプリタが実行する仮想命令連結の一例を説明するための第3の図。

【図7】同実施形態のインタプリタが実行する仮想命令連結の一例を説明するための第4の図。

【図8】同実施形態のインタプリタが実行するネイティブ手続き呼出命令への書き換えの一例を説明するための第1の図。

【図9】同実施形態のインタプリタが実行するネイティブ手続き呼出命令への書き換えの一例を説明するための第2の図。

【図10】同実施形態のインタプリタのプログラム実行方法の動作手順を説明するための第1のフローチャート。

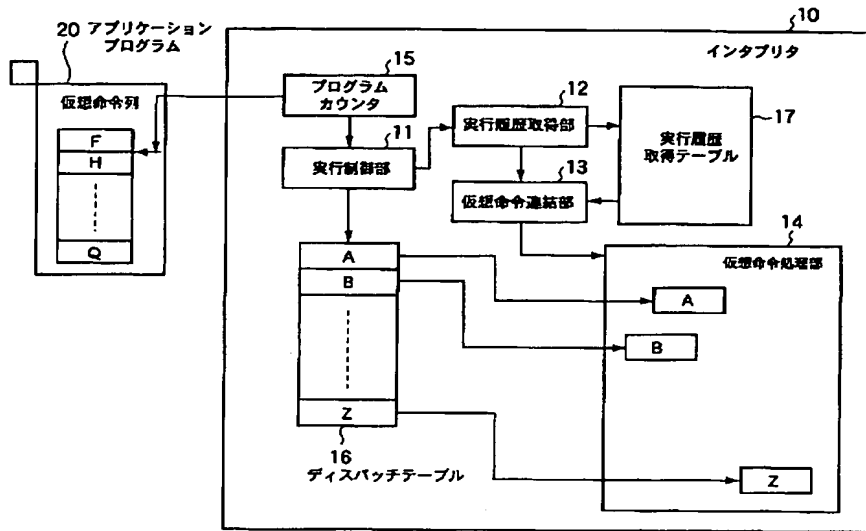
【図11】同実施形態のインタプリタのプログラム実行方法の動作手順を説明するための第2のフローチャート。

【符号の説明】

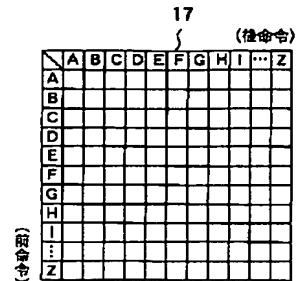
- 10…インタプリタ
- 11…実行制御部
- 12…実行履歴取得部
- 13…仮想命令連結部
- 14…仮想命令処理部
- 15…プログラムカウンタ
- 16…ディスパッチテーブル
- 17…実行履歴取得テーブル
- 20…アプリケーションプログラム

21…仮想命令列

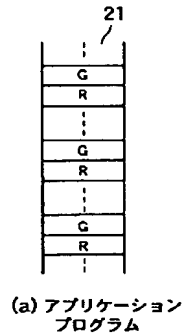
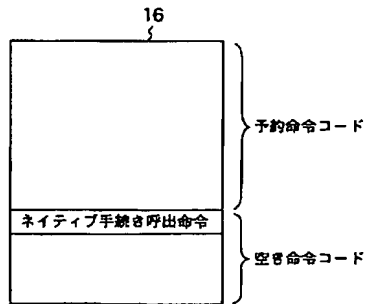
【図1】



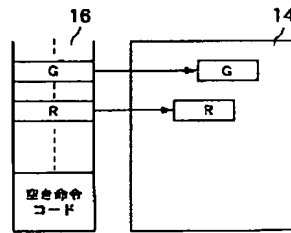
【図3】



【図2】

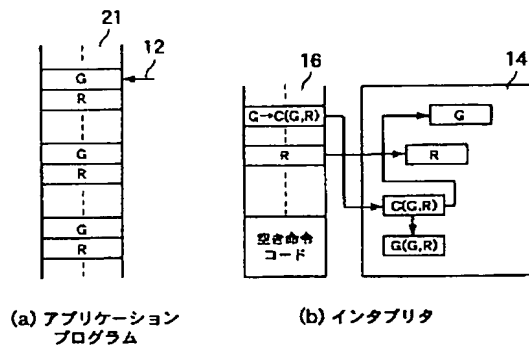


【図4】



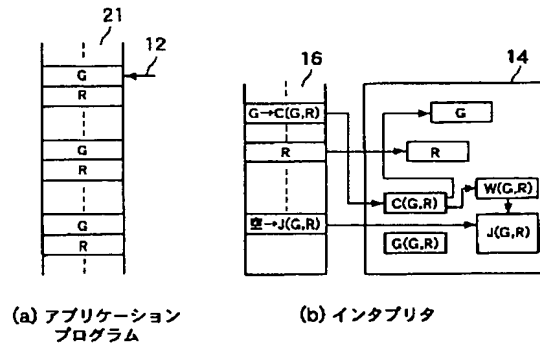
(b) インタプリタ

【図5】



(b) インタプリタ

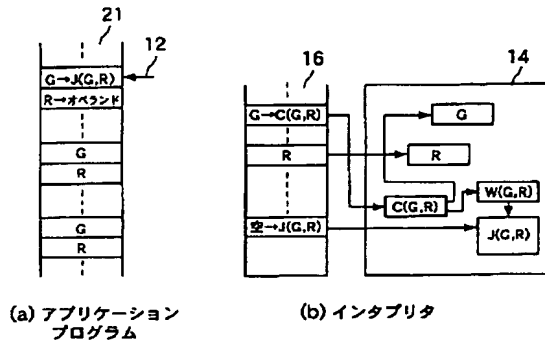
【図6】



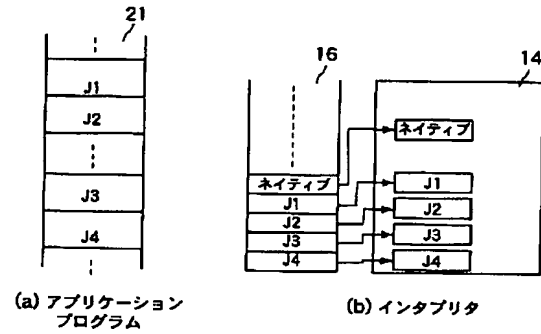
(b) インタプリタ



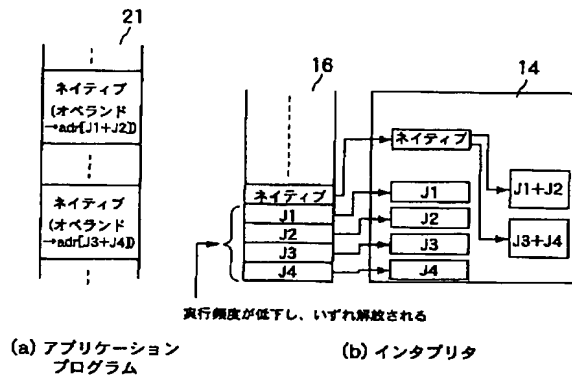
【図7】



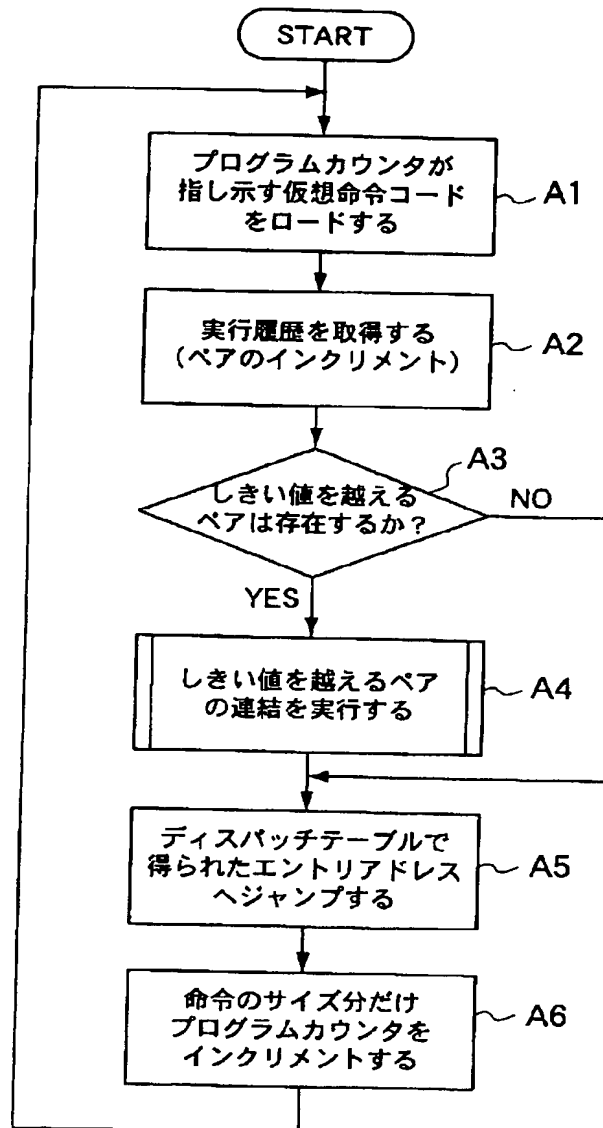
【図8】



【図9】



【図10】



【図11】

